BY GREGG ZEHR

MEMORY MANAGEMENT UNITS FOR 68000 ARCHITECTURES

Design options that speed up memory management

The Motorola 68000 family of microprocessors has spawned a whole new group of computer systems. The original 68000, with its large, linear addressing range, makes it a natural for single-user, personal graphics workstations such as the Macintosh. And multiuser systems based on the 68020 can offer computing power and speed that rival many minicomputers—often at a fraction of the cost. Not surprisingly, many of the design features for these larger systems have evolved from well-established minicomputer architectures. Memory management units, or MMUs, are one example. The MMU function came about as minicomputer designers began to include special hardware to expand the amount of addressable memory. MMUs have now become a key feature in modern computer architectures. In fact, several MMUs designed specifically for the 68000-family architecture are available (see table 1).

THEORY OF OPERATION

The MMU functions at a very low level in the computer system. Unlike a UART or other peripheral chip that attaches to the system bus and is idle most of the time, the MMU attaches directly to the CPU address bus and intercepts each CPU read or write cycle. The CPU and MMU combine to form a new functional unit. Several manufacturers have even moved the MMU onto the same silicon as the CPU, in effect declaring that you can't have one without the other.

The most important function provided by all MMU designs is the ability to relocate a program to another part of memory according to a set of pre-assigned translation rules. This relocation is done in hardware, without requiring any modification to the application software.

Before a system with an MMU runs a program, the operating system configures the MMU so that the program can be moved to and run in an available section of memory. The program then begins execution, unaware of the MMU's actions. For example, if a program has been compiled and linked with a starting location of 400 but that location is being used for some other purpose, the operating system configures the MMU hardware to convert all the program's memory references to an unused section of memory. Although the MMU is obviously useful in a system that has multiple users running separate programs, it is just as useful in a multitasking single-user system.

In a simple 68000 system that does not have an MMU (figure 1), a typical memory read cycle begins when the CPU asserts an address and address strobe (AS), and the cycle ends when the memory places data on the data bus and activates the data transfer acknowledge (DTACK) line. Assuming that the memory is very fast, the cycle can be completed in eight transitions of the clock, or 500 nanoseconds for an 8-MHz CPU.

In a 68000 system that has an MMU in series with the CPU's address bus (figure 2), for each read cycle the CPU asserts a logical address and logical address strobe (LAS). (The address and address strobe lines are now

(continued)

Gregg Zehr is a senior design engineer at Altos Computer Systems (2641 Orchard Parkway, San Jose, CA 95121). He received his M.S.E.E. from the University of Illinois and is interested in advanced computer architectures.

prefaced with the term *logical* since they are the absolute addresses from the CPU's point of view.) The MMU accepts the address and logical address strobe and then translates the logical address according to a set of translation rules into a physical address. It then asserts a physical address and a physical address strobe (PAS). (The term *physical* is used to indicate that these addresses are physically attached to the memory.) The memory again responds by putting data on the data bus and asserting DTACK.

But, as the saying goes, nothing is free. There are two penalties for attaching the MMU in series with a bus—speed and pin count. First, each memory cycle must now be slowed down while the MMU performs the translation. Second, the MMU must monitor a wide input bus and drive a wide output bus. Expect a single-chip MMU for a 68000/68010-based system to have at least 64 pins and an MMU for the 68020 to have over 120 pins. Although the cost of a device is directly proportional to the number of pins on the package, in most systems, fortunately, the cost of adding MMU hardware is less than developing a layer of software to perform similar functions.

Since the MMU operates on each memory access, it is the perfect place to add special hardware support for certain operating system functions that are not strictly related to address translation. The most important extras are memory protection, cache, and virtual memory support hooks. For example, by monitoring the three function code bits from the 68000, the MMU can divide the CPU's address space into user- and supervisor-level instruction and data areas. Thus, while you debug a program, the MMU can trap unauthorized (usually unintentional) attempts to access reserved system functions such as memory-mapped I/O or interrupt vectors. In this case, the MMU hardware ensures that a bug in a program does not hang the system.

Table 1: A *summary of memory management units.*

Device	Manufacturer	Translation	CPU Supported
68451	Motorola	Segmented	68008/00/10
68905	Signetics	Segmented/Paged	68000/10
68070	Signetics/Philips	Segmented/Paged	Integrated 68000
68910	Signetics	Demand paged	68010
68920	Signetics	Demand paged	68020
68461	Motorola	Demand paged	68010/20
68851/MMB	Motorola	Demand paged	68010/20
68851	Motorola	Demand paged	68010/20

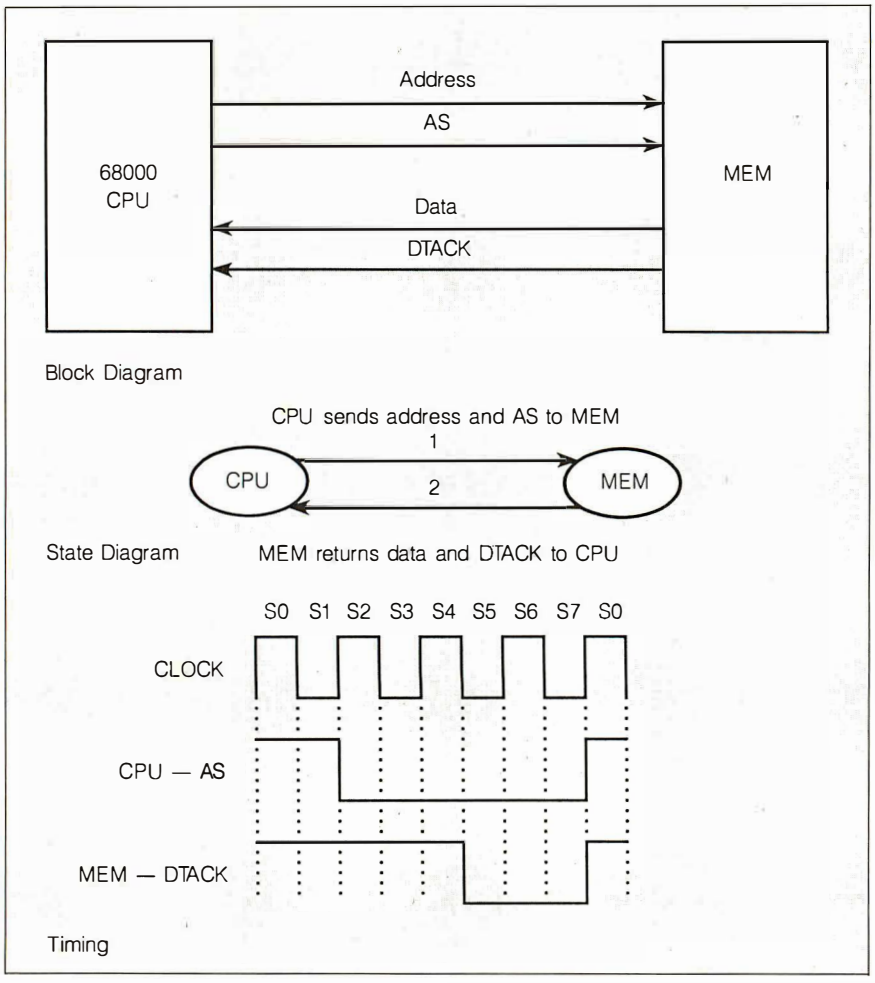

Figure 1: A 68000-*based system without an* MMU.

Paged Translation

The translation rules that an MMU uses can be classified as being either paged or segmented. Paged systems usually divide memory into equal-size pieces (pages), while segmented systems divide memory into variable-size pieces (segments). Both of these concepts first appeared in mainframe and minicomputer systems.

In a paged translation (figure 3), the MMU divides the logical addresses into two parts; the upper bits are called the segment number and the lower bits are called the page index. The page index, which determines the page size, is passed directly through the MMU unmodified. The segment number is used as an address into a segment table. The data from the segment table is called the page address and forms the upper part of the physical address. Logically then, a memory location is described by a

13-bit offset into one of 2048 pages. Physically, memory is divided into 2K pages with a fixed size of 8K bytes.

Although several popular 68000/68010 systems have been built by simply implementing the segment table with high-speed static RAM, such an approach does not fit into 32-bit 68020 designs very well. If the lowest 12 lines are used for the page index, there are 20 lines left. This implies that the segment table RAM must hold 1 megabyte of page numbers.

Since pages have a fixed size, this type of translation is susceptible to internal memory fragmentation. This means that some segments will likely include memory that is unused. For example, suppose that a program needs 1K byte of storage for its data. When run, the system assigns the program one 4K-byte segment. The other 3K bytes become a memory fragment that cannot be used by any other program. Most paged systems include at least two levels of translation and a smaller page size that reduces such internal fragmentation.

A simple trick, however, can increase the capabilities of this approach. The segment table RAM can be wider than the segment number to provide additional control bits, and from the physical address bus, these control bits cannot be distinguished from normal 68000 control lines. So these extra bits can be used as address lines, and in fact this technique has worked to extend the addressing capability of CPUs ranging from the 6502 to the PDP-11. Other uses for these bits include memory protection attributes, virtual memory paging indicators, and cache inhibit mark bits.

Segmented Translation

In theory, segmented translation should be more efficient since most memory requests are not integer multiples of some fixed-size page. The upper bits of the address are called the segment number and the lower bits are called the segment displacement or offset (figure 4). The segment number is used to address a table of descriptors. The descriptor includes a base address, which is the starting address of the segment in physical memory. The descriptor also includes the length of the segment. The segment offset should be smaller than the length; if it is not, the memory cycle is aborted and an error is indicated. Assuming there is no error, the translation is completed by arithmetically adding the segment offset to the base address. Physical memory can now be divided into 256 variable-size segments. Each segment can be from 1 to 64K bytes long.

Although variable segment size allows memory allocation to fit memory requests better, it leads to another problem called external fragmentation. This problem, which is unique to segmented MMUs, occurs when variable-size segments leave holes in physical memory that are too small for practical use. Several algorithms have been developed to simplify allocation in segmented systems and are described by Baer and Knowlton (see the Bibliography).

Demand-Paged Translation

As CPU buses become wider, the amount of memory required to store page tables or segment descriptors becomes larger. This in turn increases the cost of the MMU and the overhead associated with task switching.

(continued)

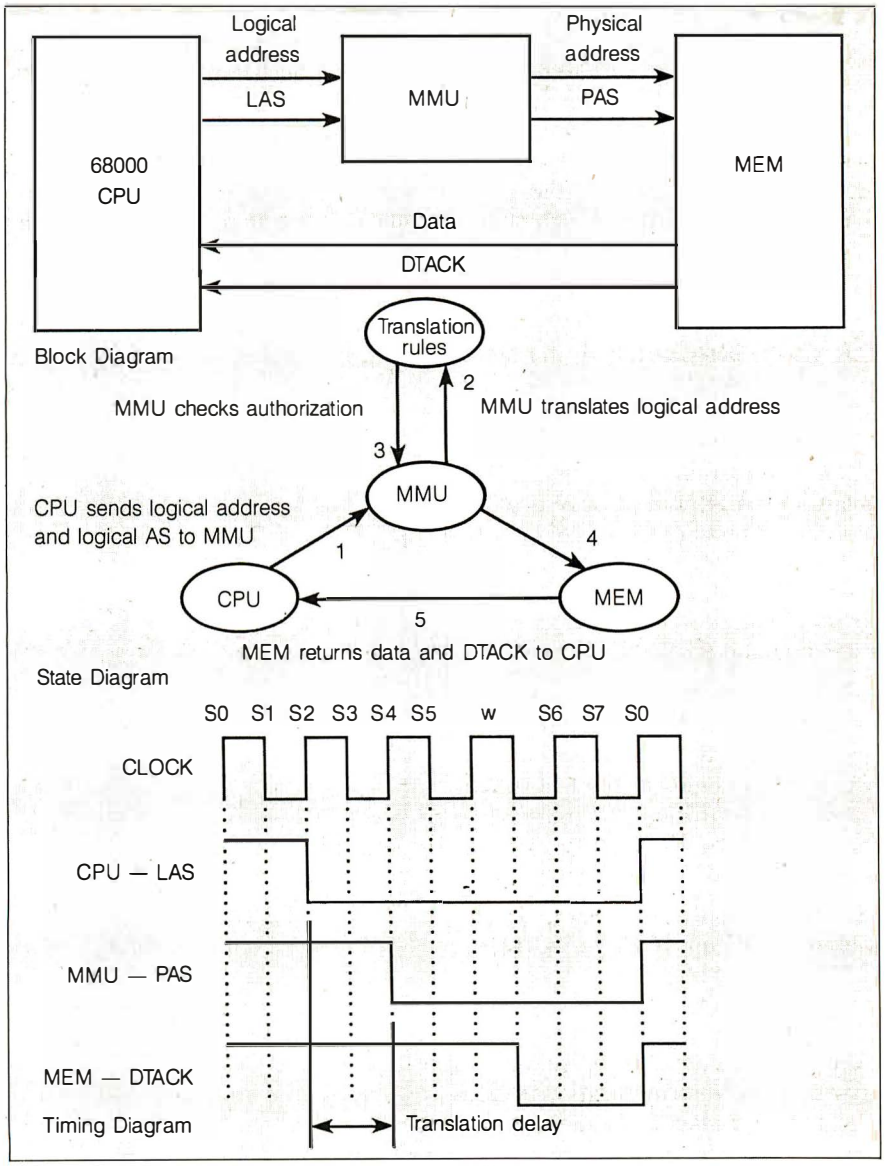

Figure 2: A 68000-*based system with an* MMU.

The demand-paged MMU provides support for 32-bit microprocessors by allowing translation tables to be stored in main memory (figure 5). When the system initializes, the CPU writes the translation tables into main memory and then tells the MMU where they are by writing a pointer into a control register. The MMU includes bus control logic that allows it to search the tables and find the correct translation information. To avoid searching the tables for each translation, the MMU maintains a buffer of recently used translation information in a small cache memory called a translation lookaside buffer (TLB). Thus, once the tables are established, the MMU can translate any logical address without advance warning from the CPU (translation on demand). This arrangement also means that the MMU will only take time to fetch those descriptors that are actually used by a program; in a timeshared, multiuser system this method is usually more efficient than loading all the descriptors each time the program runs.

When a demand-paged MMU finds the translation information in its TLB, it translates the logical address into a physical address. If the translation information is not in the TLB, the MMU must back the CPU off the bus while it searches the translation tables in the main memory. This search process is referred to as a table walk. At the end of its table walk, the MMU writes the new descriptor into its TLB and tells the CPU to retry the access.

Although the demand-paged MMU provides an elegant solution to a difficult problem, the table walk process is slow. A typical table walk will cost the CPU 20 or more wait states. This means that the performance of the MMU is governed by the percentage of times that it finds the needed information in the TLB—the TLB hit rate. If the CPU includes an efficient cache memory, the TLB hit rate may very

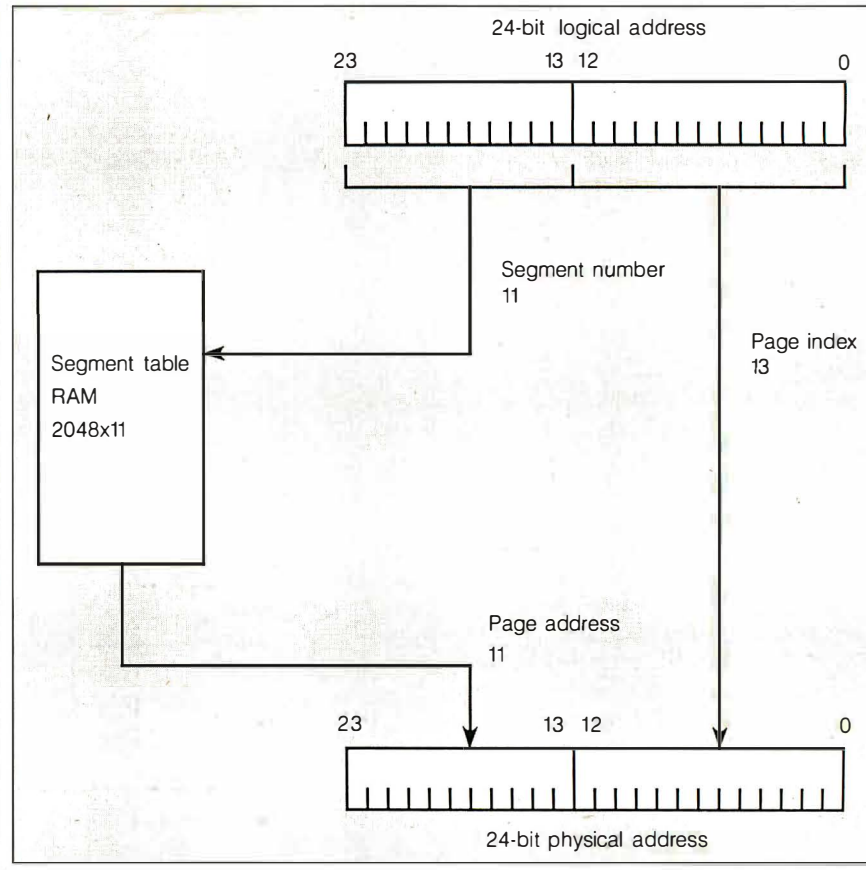

Figure 3: *In a paged translation the upper bits are called the segment number and the lower bits are called the page index.*

well become the factor that limits system performance.

If you plan to include an MMU in your next design, you must match the CPU and MMU combination with the overall system architecture and cost. For each possible MMU design alternative you must consider hardware and software issues. The most important hardware issues include how to minimize translation delay, how the MMU should signal error conditions to the CPU, and how to reduce hardware overhead related to a software task. Of course, the nature of these issues depend on whether you are using a 68000, 68010, or 68020.

In typical systems, a discrete paged MMU will support a simple operating system or real-time executive in a small single-user or embedded control system. Segmented systems have been used in large computers for many years, but the advantages are probably not worth the additional complexity in a small system. The demand-paged memory system provides the best features of both paged and segmented systems and has become the standard for multiuser UNIX machines.

MMU Design Options

Given an understanding of the MMU's theory of operation and the system design considerations, there remain the actual design implementation options. The first and most obvious option is to not use an MMU at all. That's exactly the design decision made for the Apple Macintosh, the Commodore Amiga, and the Atari 520ST. Although the graphics interface used by these machines, which includes multiple windows and desk accessories, may give the impression the systems perform multitasking quite naturally, none of these systems includes any MMU hardware. Instead, they place the burden of memory management on application software. In each case

(continued)

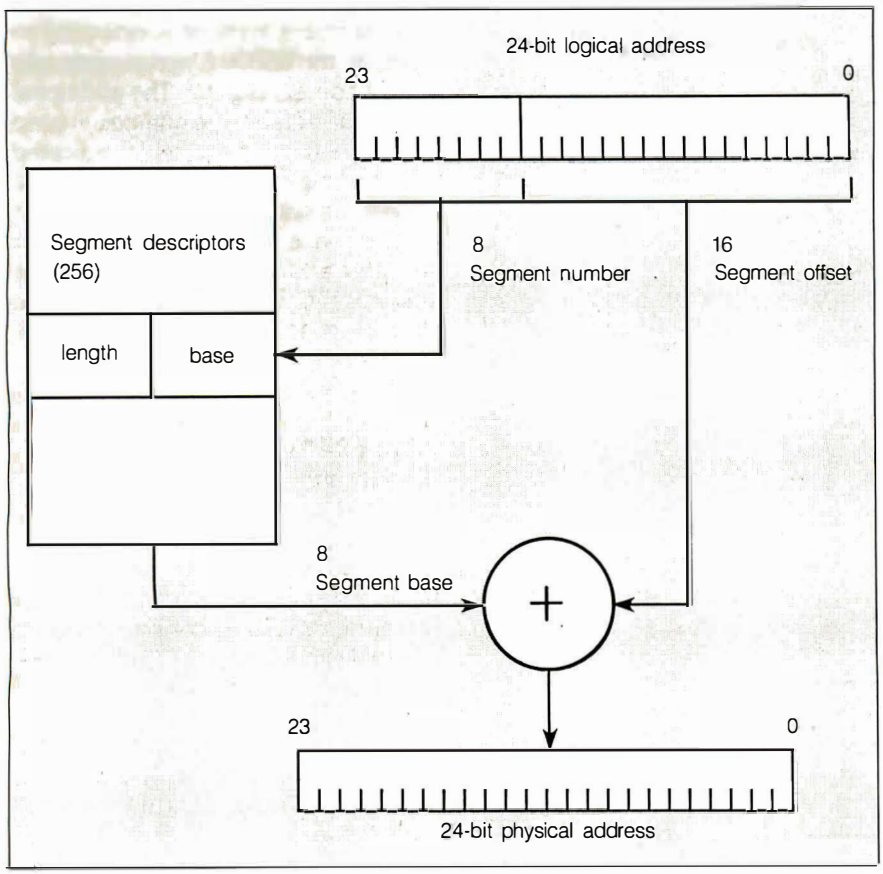

Figure 4: *In a segmented translation the upper bits are called the segment number and the lower bits are called the segment displacement or offset.*

the designers have provided a real-time executive with a set of low-level interface routines and a complex set of so-called gentlemen's agreements to provide multitasking. For example, an application running on one of these machines must understand when and how to call low-level routines for tasks such as memory allocation. It is also up to the application to decide what to do if those routines cannot allocate the requested memory.

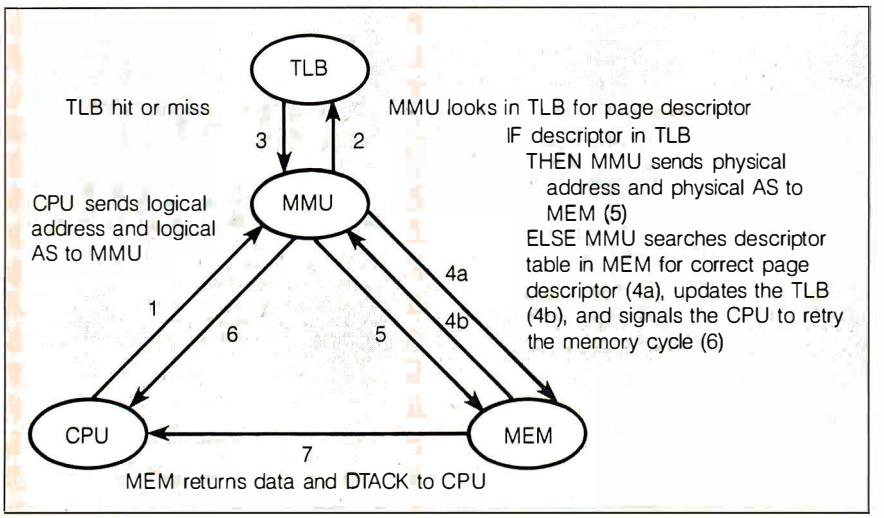

Figure 5: *Demand-paged translation state diagram.*

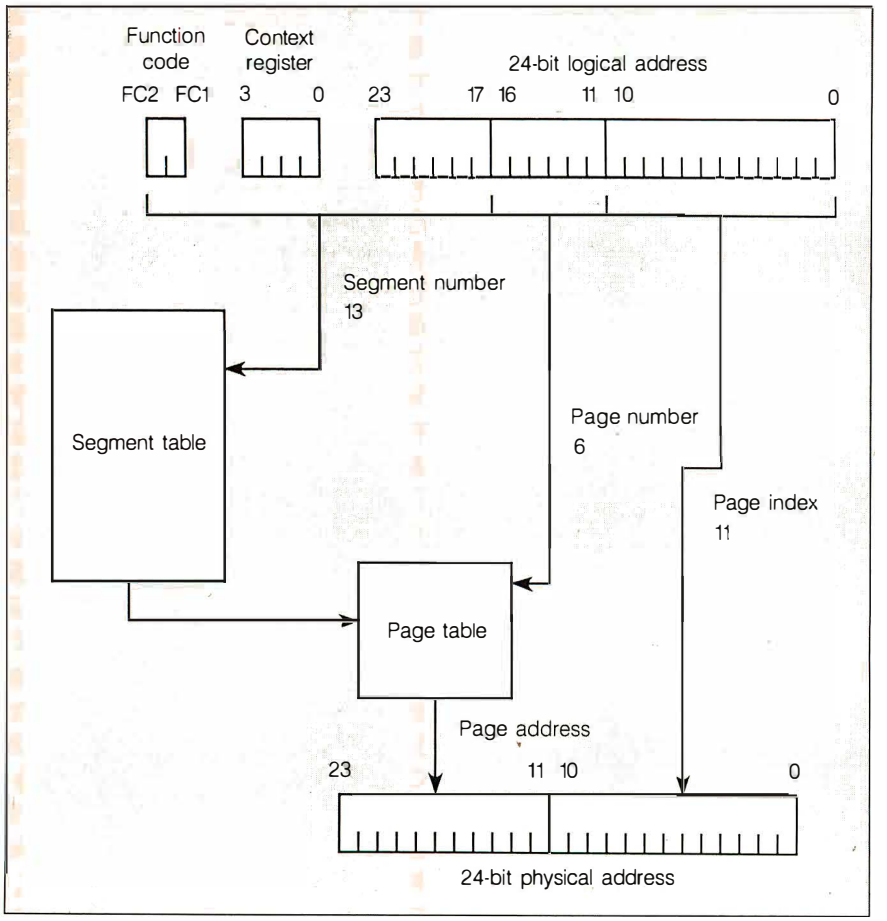

Figure 6: *Two-level paged translation.*

Also, when a program runs on one of these machines, it has access to all system resources, and a programming error can easily write over any of the other programs in memory including the operating system. This is usually a fatal situation to the system, requiring a power reset. Even a modest amount of MMU hardware could improve the performance and reliability of these machines by reducing the amount of memory management the operating system has to perform and by providing memory protection in hardware.

A paged MMU can also be built using discrete logic and high-speed static RAM to hold the segment table. This approach was very common in early 68000-based multiuser systems. The basic paged-translation concept is usually extended to two levels (figure 6). In this approach, the logical address is divided into three fields. The segment number is extended to include the 68000 function code bits and a context register. The additional field, called the page number, is used to address the page table indicated by the segment table output. Most operating systems set up the segment tables once, then use the context register and page tables to allocate memory for each task. The page tables are small enough that they can be paged to main memory when a task switch occurs.

THE 68451 MMU

Shortly after the first 68000 CPU chip made its debut, the Motorola 68451 appeared. It was, in fact, one of the first monolithic MMUs available to system designers. The 68451 is a segmented MMU that comes packaged as a 64-pin DIP (figure 7). It includes 32 segment descriptors that partition memory into variable-size segments. Each of these descriptors also includes an 8-bit status register that provides support for a virtual memory architecture.

There are several serious limitations with this device, however. The biggest problem is that 32 descriptors are not enough. The 68451 includes special

lines that allow several chips to be chained together to expand the number of descriptors, but since the MMU still costs almost twice as much as the CPU, this is an expensive option. The 68451 is also relatively slow. Typically, translation requires more than one wait state (especially in a multi-MMU system), and if a task switch requires CPU intervention (and most do), the overhead is greater than that of a simple paged system. The 68451 also lacks support for CPU cache memory or the 32-bit 68020. Since all of these problems have been resolved with a new Motorola MMU chip, the 68851, the 68451 will probably not be used in many new designs.

If you are porting an operating system to a machine that does use this device, you should consider using the binary buddy memory allocation algorithm as described by Knowlton. This algorithm should allow you to take advantage of the variable segment size while reducing fragmentation and operating system memory allocation overhead.

THE 68905 BMAC

The 68905 basic memory access controller (BMAC) is the first in a series of ambitious announcements by Signetics and its parent company, Philips. The BMAC integrates MMU and cache control functions for

(continued)

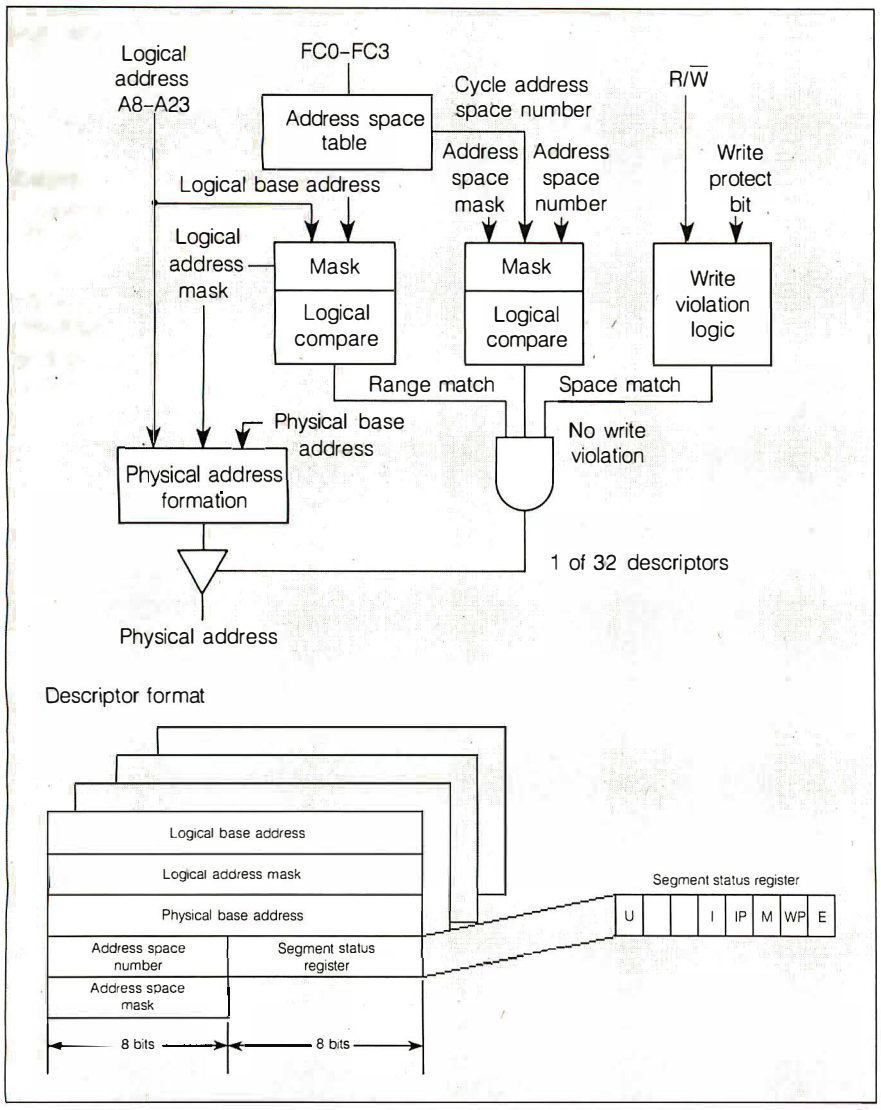

Figure 7: 68451 MMU.

68000/68010-based systems into a single 84-pin grid array package (figure 8). You can program the BMAC to perform segmented or paged translation.

Most MMUs that support paging data between hierarchies of memory allow two levels that are usually dedicated to primary memory (RAM) and secondary memory (disk). The BMAC also supports a third level, local memory (RAM). Although most operating systems do not currently support this third level, local memory could be used to provide improved performance in a multiprocessor system. Using this local memory would provide fast access to private data structures.

The BMAC also provides support for a logical bus cache memory. Placing the cache on the logical bus allows translation and cache searches to occur in parallel, but in order to avoid cache coherency problems, care should always be taken to flush the cache at each task switch. Although a logical cache is fast, it may not, however, be transparent to the operating system.

Signetics has also announced the 68910 and 68920 memory access controllers, or MACs, that extend the BMAC design by including a microcontroller that effectively provides demand-paging capability for 68010 and 68020 systems.

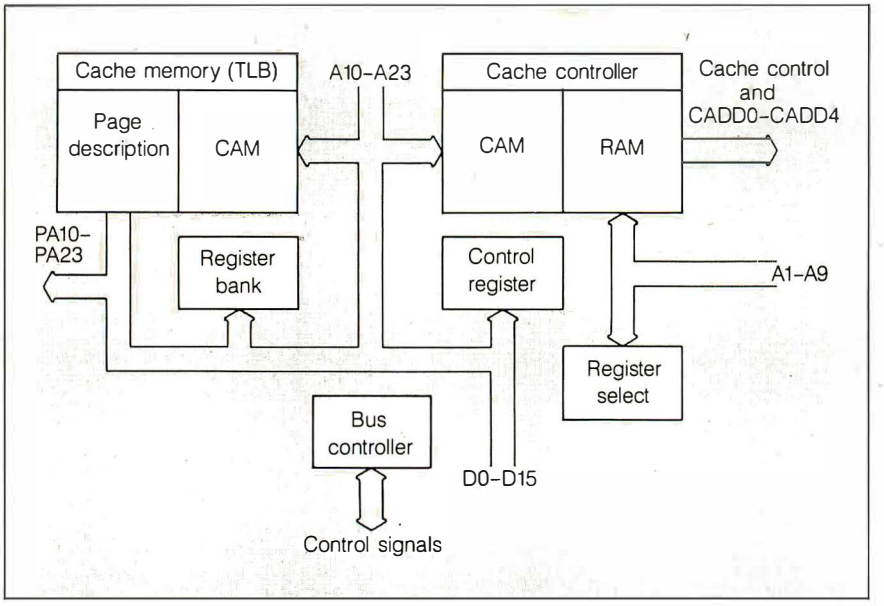

Figure 8: 68905 *block diagram.*

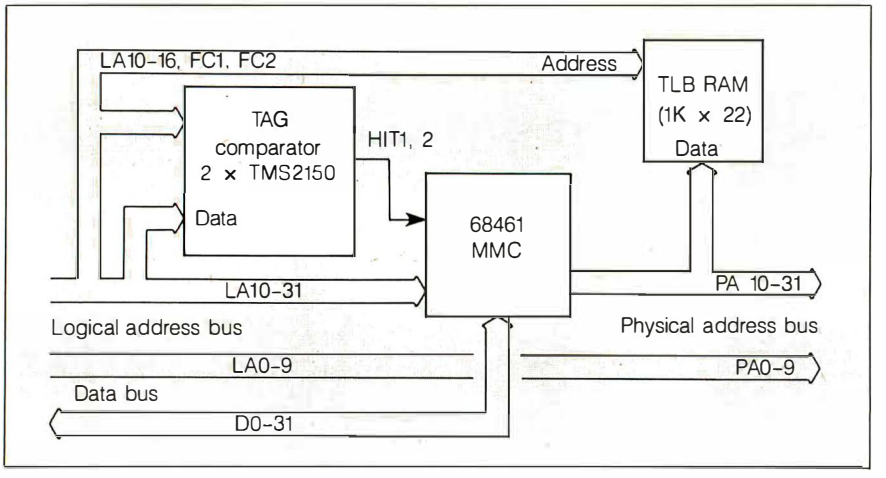

Figure 9: *Block diagram of* MMU *using* 68461.

THE 68461 MMC

Shortly after introducing the 32-bit 68020 CPU chip, Motorola announced its plans to develop a demand-paged virtual MMU, which would support multitasking, multiuser environments such as UNIX. Unfortunately, the new MMU chip was not ready in time to be shipped with the first CPUs. Recognizing the need for MMU support, Motorola made the 68461 memory management controller (MMC) available as an interim solution—until the single-chip 68851 paged MMU is available.

The 68461 is fast; it's built with Motorola's 2800-series bipolar gate array, and it can translate a 16-MHz 68020 access in one wait state. The MMC is housed in a 147-pin grid array package, which requires a heat sink. The MMC does not include everything required to implement a demand-paged MMU. To use this device, you must use external logic to implement the TLB function. A single-set-associative TLB can be built with 15 or 16 external chips (figure 9). Even this simple TLB architecture, however, offers a hit rate in the UNIX environment of better than 90 percent, which is high enough to provide good system throughput.

An MMU incorporating the MMC can provide demand-paged memory support for either the 68010 or the 68020. It includes the extra control bits that are required for memory protection, virtual memory, and CPU cache memory functions. It maintains its translation descriptors in a tree structure in main memory. The translation process divides the logical address into three fields, which are used to search three levels of descriptors (figure 10). Limit fields at each level of the table reduces the total amount of RAM needed to hold the descriptors. Yet a typical system requires about 128K bytes for the MMU. Protection bits at each level of the table can provide read and write access protection based on the function codes. For example, you can configure a page to allow supervisor read

and write access while a user mode write will generate an error.

THE 68851 PMMU

The Motorola 68851 paged MMU, or PMMU, provides complete demand-paged MMU support in a single chip. This advanced design includes a 64-entry fully associative TLB that is more efficient than the single-set design of the 68461 MMC. Since the PMMU attaches to the 68020's coprocessor interface, its registers are extensions to the existing programmer's model of the CPU. The CPU/PMMU combination adds new MMU instructions to the existing 68020 instruction set. While all the other devices discussed here decode the MMU's control registers as memory-mapped I/O, this coprocessor approach integrates the PMMU into the programming environment. For example, a single instruction allows a conditional branch based on the condition of the PMMU status register.

The PMMU's translation mechanism is similar to the MMC, but the PMMU offers more flexibility. The PMMU page sizes can range from 256 to 32K bytes, and page tables are not fixed at three levels. The PMMU can partition the logical address into one to four fields, each of which serves as an index to the table at that level.

PMMU hardware includes arbitration logic for both the logical and the physical bus. A separate pointer register is provided for an alternate logical bus master, such as a DMA controller. In a multiprocessor environment, PMMU's can share descriptor tables in main memory, reducing storage requirements. The PMMU offers full support for system functions such as virtual memory, cache memory, and a floating-point coprocessor.

Besides using the CPU function code bits for memory protection, the PMMU adds up to eight levels of access authorization. This concept is also extended into the 68020 call module (CALLM) and return from module (RTM) instructions so that authorization can be verified at the subroutine level.

For the faint of heart, Motorola also offers the 68461 and discrete TLB

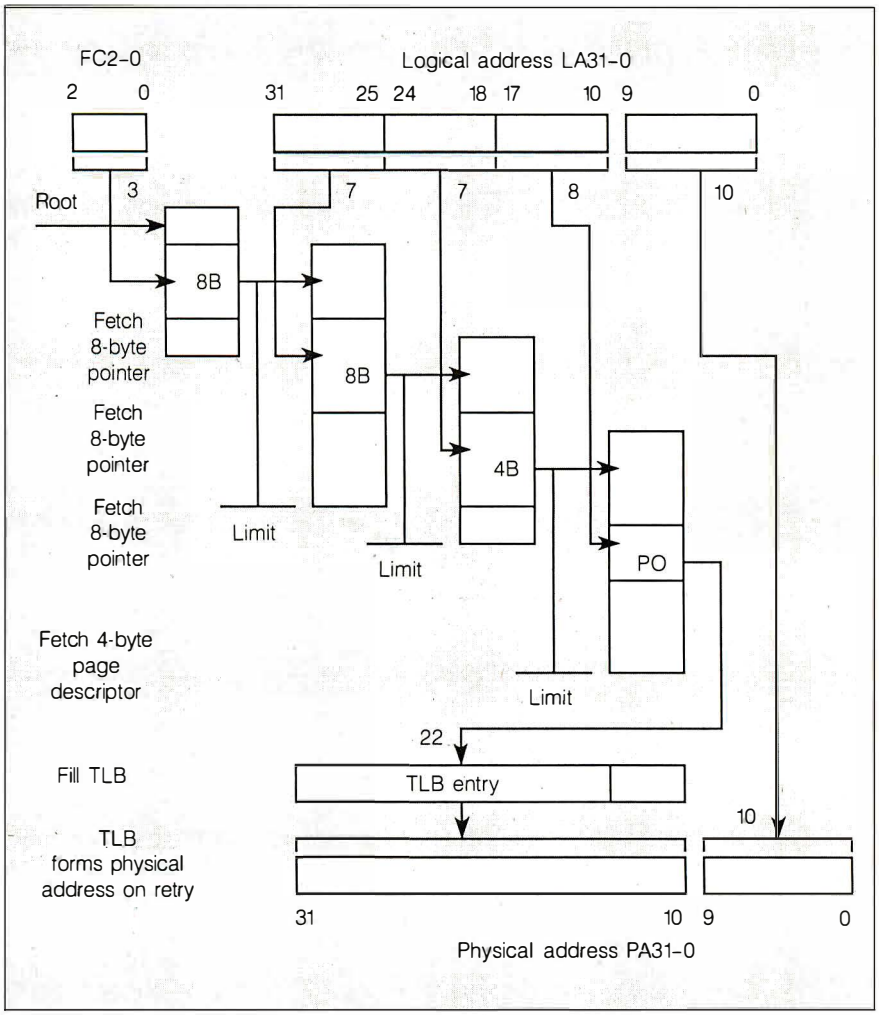

Figure 10: *Demand-paged translation.*

already assembled on a printed circuit board, which is pin-for-pin compatible with the 68851. You can plug this board-level product, the 68KVMMB851, into your next 68020 design to provide MMU support until the 68851 is available in production quantities.

FUTURE TRENDS

It's difficult to determine which has advanced more rapidly, the microprocessor or the MMU. Certainly the supermicrocomputers available today depend on the MMU just as much as the microprocessor to provide high performance for a lower-than-ever cost per user. If history is any indication, IC manufacturers will continue to integrate more and more system functions onto silicon.

Integrated units that combine the 68000 CPU and a simplified 68920 MMU in a single device, such as the recently announced Signetics/Philips 68070, are sure to abound in the future. The advantages of putting the CPU and MMU on the same silicon include faster translation, lower pin count (and therefore cost), and improved software portability. Moreover, by offering silicon that can simplify the layer of software required for multitasking, this device is sure to find its way into the next generation of mouse-and-windows machines. ■

BIBLIOGRAPHY

Baer, Jean-Loup. *Computer Systems Architecture.* Rockville, MD: Computer Science Press, 1980.

Knowlton, Kenneth C. "A Fast Storage Allocator." *Communications of the ACM,* vol. 8 (October 1965), pages 623–625.